LLM-Cure: LLM-based Competitor User Review Analysis for Feature Enhancement

MARAM ASSI, Queen's University, Canada
SAFWAT HASSAN, University of Toronto, Canada
YING ZOU, Queen's University, Canada

The exponential growth of the mobile app market underscores the importance of constant innovation and rapid response to user demands. As user satisfaction is paramount to the success of a mobile application (app), developers typically rely on user reviews, which represent user feedback that includes ratings and comments to identify areas for improvement. However, the sheer volume of user reviews poses challenges in manual analysis, necessitating automated approaches. Existing automated approaches either analyze only the target app's reviews, neglecting the comparison of similar features to competitors or fail to provide suggestions for feature enhancement. Given their strong natural language understanding capabilities, Large Language Models (LLMs) offer an effective solution for extracting insights from vast amounts of user reviews and generating meaningful suggestions. To address these gaps, we propose a *Large Language Model **(LLM)**-based **C**ompetitive **U**ser **R**eview Analysis for Feature **E**nhancement)* (*LLM-Cure*), an approach powered by LLMs to automatically generate suggestions for mobile app feature improvements. More specifically, *LLM-Cure* identifies and categorizes features within reviews by applying LLMs. When provided with a complaint in a user review, *LLM-Cure* curates highly rated (4 and 5 stars) reviews in competing apps related to the complaint and proposes potential improvements tailored to the target application. We evaluate *LLM-Cure* on 1,056,739 reviews of 70 popular Android apps. Our evaluation demonstrates that *LLM-Cure* significantly outperforms the state-of-the-art approaches in assigning features to reviews by up to 13% in F1-score, up to 16% in recall and up to 11% in precision. Additionally, *LLM-Cure* demonstrates its capability to provide suggestions for resolving user complaints. We verify the suggestions using the release notes that reflect the changes of features in the target mobile app. *LLM-Cure* achieves a promising average of 73% of the implementation of the provided suggestions, demonstrating its potential for competitive feature enhancement.

CCS Concepts: • **Software and its engineering → Software maintenance tools**.

Additional Key Words and Phrases: LLM, Mobile applications, User reviews, Feature Enhancement, Competitor analysis

ACM Reference Format:
Maram Assi, Safwat Hassan, and Ying Zou. 2024. LLM-Cure: LLM-based Competitor User Review Analysis for Feature Enhancement. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (April 2024), 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The mobile application (app) market is experiencing explosive growth, with global downloads reaching a staggering 257 billion in 2023 [11]. This surge in app adoption has fostered a highly competitive environment among apps within the same categories that offer similar functionalities,

Authors' addresses: Maram Assi, maram.assi@queensu.ca, Queen's University, Kingston, Ontario, Canada; Safwat Hassan, safwat.hassan@utoronto.ca, University of Toronto, Toronto, Ontario, Canada; Ying Zou, ying.zou@queensu.ca, Queen's University, Kingston, Ontario, Canada.

ACM 1049-331X/2024/4-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

i.e., competitors. For instance, WhatsApp[1], leading the messaging app category with 51 million monthly downloads, competes with at least ten other apps, each having millions of downloads [10]. A similar competition is evident in the video conferencing category, where Zoom[2] and Skype[3] are key players[15, 44]. During the pandemic, Zoom swiftly adapted to user demands by optimizing its platform for large-scale virtual meetings and enhancing security features, while Skype struggled to keep pace [47, 55]. To stay relevant and competitive, developers must rapidly respond to user needs. User reviews contain rich information, such as feedback, dissatisfaction and suggestions regarding the user experience of the usage of mobile apps. These reviews offer developers critical insights into areas for improvement and opportunities for feature enhancements [2, 45, 50, 57, 62]. To stay competitive, developers need to learn from their competitors' behaviours to maintain a competitive edge [54]. Competitor user review analysis involves comparing user feedback, ratings, and reviews of competing mobile applications to identify strengths and weaknesses relative to competitors. By analyzing user reviews from competing apps, developers can uncover insights into features that address unmet needs, potentially giving their app a significant advantage.

Given the sheer volume of reviews [25], it is challenging to analyze user reviews manually. Practitioners need an automated feedback analysis process [57]. Hence, researchers propose automated approaches to filter informative reviews [12, 24], summarize user reviews [22, 59] and extract features from user reviews [14, 32, 33, 53, 59]. While existing research has been conducted on automated user review analysis, only a limited number of studies focus on competitor user review analysis, where reviews across competing apps are compared [7, 27, 34, 42, 51, 54, 61]. Recent advancements in Large Language Models (LLMs) have demonstrated their capabilities across various natural language processing (NLP) tasks, including text generation, translation, summarization, and question answering [40]. Although LLMs offer promising applications for mobile app review analysis, current LLM-based research primarily focuses on tasks, such as sentiment analysis [48, 66], aspect extraction [64], analyzing multilingual reviews [63] and accessibility-related reviews [19].

Researchers have explored various approaches to conduct competitor user review analysis [7, 14, 34, 36, 51, 54]. However, existing work presents some limitations. First, the existing approaches often generate an overwhelming number of fine-grained features [14, 51, 54] due to comparing the apps' features based on word pairs, making it hard to conduct competitor user review analysis with thousands of features [52]. Second, competitor user review analysis is only conducted by identifying explicit expressions of comparison (e.g., *"Zoom's screen sharing is way smoother than Skype's"*) [34]. However, users often provide feedback without explicitly naming competitors, making it challenging to extract meaningful comparisons. Implicit insights refer to the underlying sentiments, feature expectations and areas of dissatisfaction that can be inferred even when direct comparisons are not stated.

Third, existing work on competitor user review analysis [7] offers only feature rating comparisons lacking the ability to suggest concrete improvements for specific features based on competitors' user feedback.

To address the limitations of existing work, we propose an ***LLM**-based **C**ompetitive **U**ser **R**eview Analysis for Feature **E**nhancement (LLM-Cure). LLM-Cure* automatically generates suggestions for mobile app feature improvements by leveraging user feedback on similar features from competitors. *LLM-Cure* operates through two phases. In the first phase, it leverages its large language model capabilities to extract and assign features to user reviews. In the second phase, it curates underperforming features among those identified in the first phase for the target app and suggests potential

[1]https://play.google.com/store/apps/details?id=com.whatsapp
[2]https://play.google.com/store/apps/details?id=us.zoom.videomeetings
[3]https://play.google.com/store/apps/details?id=com.skype.raider

improvements for specific complaints by leveraging highly rated similar features in competing apps. Specifically, *LLM-Cure* leverages large language models such as *Mixtral-8x7B-Instruct-v0.1* and *GPT-4o mini* to extract and categorize features in a more semantically meaningful manner, reducing noise and eliminating overly granular feature sets that limit existing work [14, 50]. Unlike prior work, *LLM-Cure* goes beyond explicit comparisons by analyzing user complaints and retrieving highly rated competitor reviews to infer missing features and identify potential enhancements. *LLM-Cure* leverages the semantical categorization of features to conduct the comparison. Additionally, *LLM-Cure* generates actionable suggestions by learning from competitor features and providing developers with concrete recommendations rather than just comparative ratings. By integrating these capabilities, *LLM-Cure* enhances the effectiveness of competitor user review analysis, enabling more informed decision-making for feature improvements.

To evaluate the effectiveness of our proposed approach, we conduct an empirical study on 1,056,739 reviews of 70 popular mobile apps from the Google Play store belonging to 7 categories. We evaluate the ability of *LLM-Cure* to (1) accurately assign features to user reviews and (2) offer the developers targeted suggestions for improving the features of their apps based on a specific complaint. We structure our study along by answering the following research questions (RQs):

RQ1: How effective can LLMs be in extracting features from user reviews?
Automatically extracting features from user reviews provides developers with an efficient way to gain valuable insights into the strengths and weaknesses of mobile app features. In this RQ, we evaluate the ability of *LLM-Cure* to automatically extract features from user reviews. We show that *LLM-Cure* achieves an average F1-score of 85%, an average recall of 84% and an average precision of 86% in assigning features to user reviews, outperforming the state-of-the-art approach by 7%, 9% and 4% in F1-score, recall and precision respectively on average.

RQ2: Can LLMs leverage categorized user reviews to generate suggestions for feature improvements?
Automatically generating feature improvement suggestions from competitor user reviews allows developers to address specific complaints and stay competitive in the market. By leveraging the extracted features from user reviews, *LLM-Cure* can pinpoint underperforming features and provide actionable suggestions for enhancement. To validate the suggestions generated by *LLM-Cure*, we cross-reference the suggestions with the release notes, verifying if similar improvements have been implemented in the subsequent releases. We find that 73% of the suggested enhancements by *LLM-Cure* are implemented in the release notes.

The main contributions of our work are as follows:
(1) We propose an LLM-based approach, *LLM-Cure*, for automatically suggesting feature enhancements through competitor user review analysis.
(2) We develop an efficient method for extracting and assigning features to user reviews, surpassing state-of-the-art approaches by an average of 7% in F1-score.
(3) We conduct an empirical study on 1,056,739 user reviews of popular apps from the Google Play Store to evaluate the effectiveness of *LLM-Cure* in providing suggestions for feature enhancements. *LLM-Cure* achieves a promising 73% Suggestions Implementation Rate.

Paper organization. The rest of the paper is structured as follows. Section 2 provides background about LLMs. Section 3 presents the overall proposed approach. Section 4 shows our experiments and describes the experimental setups, and results of our research questions. Section 5 describes

the possible threats of this study. Section 6 discusses the related work. Lastly, Section 7 wraps up the study and explores future research directions.

2 Background and Motivation

Large Language Models. Pre-trained Large Language Models (LLMs) are deep neural networks that have undergone extensive training on large text data that have enabled them to learn complex patterns and structures of language [67]. In the realm of software engineering, LLMs have gained significant attention and adoption for various apps [29], including code generation [18], code repair [21], and documentation generation [65]. Hence, LLMs offer promising avenues for automating software development processes and enhancing developer productivity. Although LLMs are primarily designed for generating text, their output can be influenced by specific instructions communicated through prompts [56].

Prompting. Prompting is a technique used to communicate expectations and guide the LLM's vast knowledge and capabilities towards achieving a specific goal [68]. Prompting involves providing instructions to LLMs to guide their generation process and elicit specific types of responses, enforce constraints, or guide the model toward certain stylistic elements. In the software engineering realm, prompting can be used to elicit code snippets, documentation, or other relevant text based on user requirements.

In-Context Learning and Few-Shot Learning. Leveraging pre-trained models, i.e., LLMs, for downstream tasks often requires further fine-tuning on domain-specific labeled data [30]. However, fine-tuning LLMs for specific tasks can be computationally expensive and resource-intensive, requiring substantial amounts of task-specific annotated data [9]. Hence, in-context learning and few-shot learning offer a powerful alternative [17]. In-context learning allows the LLM to adapt within a single interaction, using some initial information, i.e., context. Few-shot learning consists of exposing the model to a few examples (i.e., "*few shots*") to make the model effective for a specific task. For instance, for sentiment analysis in user reviews where the goal is to classify reviews as positive or negative, few-shot learning consists of giving the LLM a few examples for each sentiment, i.e., reviews with associated sentiment. The LLM then uses its existing knowledge and these few examples to categorize new reviews based on the context provided.

Retrieval Augmented Generation. LLMs can suffer from hallucination and generate irrelevant or inaccurate responses [31]. This can occur due to limitations in their training data or the lack of clear context in the prompt. Retrieval-Augmented Generation (RAG) addresses the issue of hallucination by integrating information retrieval techniques into the generation process [39]. Thus, the RAG retrieves relevant knowledge from external sources based on the input context, augmenting the model's understanding. For instance, in the realm of user reviews, RAG can be employed to enhance the quality of review summarization. The RAG model augments the user reviews with additional information extracted from external sources, such as sentiment analysis scores, key phrases or keywords, to generate concise and informative summaries of user reviews.

Motivation for Using LLMs in Competitor User Review Analysis. The increasing amount of user feedback generated for mobile apps makes it increasingly challenging to manually extract meaningful insights and actionable suggestions for feature improvements. Traditional methods for analyzing user reviews often fall short in handling this complexity due to issues such as the need for feature extraction from large volumes of unstructured text and the challenge of synthesizing feedback into actionable insights. LLMs provide a unique advantage in this context of processing

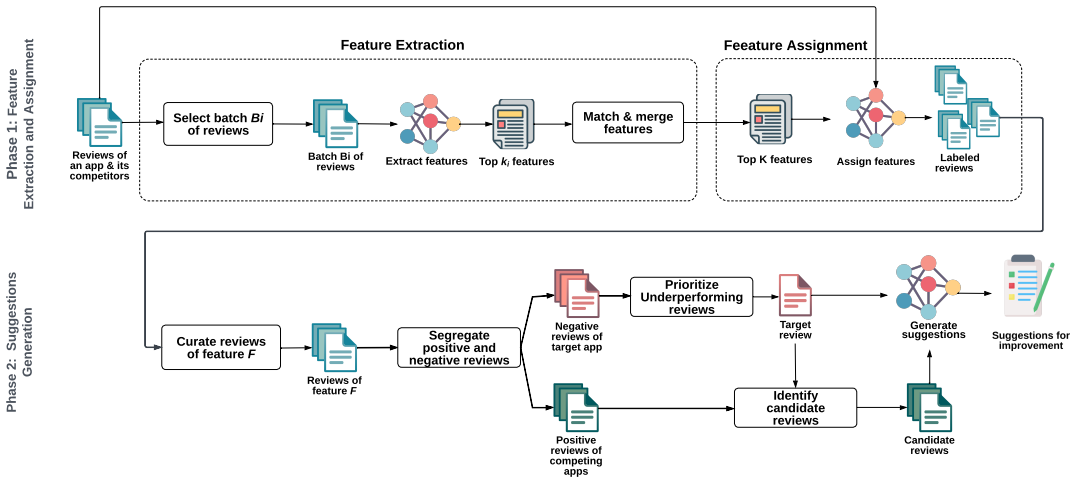

Fig. 1. Overall approach of *LLM-Cure*

a large amount of unstructured text due to their strong natural language understanding and generation capabilities. LLMs can process and analyze large-scale textual data efficiently, extracting key insights from user feedback that is context-aware. By leveraging LLMs, it is possible to not only automate the process of identifying user concerns and categorizing features, but also suggesting improvements—a task that traditionally requires significant manual human effort. This makes LLMs particularly well-suited for addressing the challenges associated with competitor user review analysis, as LLMs can handle both the volume and complexity of user feedback while providing rich, contextual insights that support actionable recommendations for mobile app development. The ability of LLMs to adapt through prompting, in-context learning, and few-shot learning also allows them to perform domain-specific tasks (for every mobile app category) with minimal retraining, further increasing their utility where rapid adaptation to new tasks and datasets is essential. Additionally, when combined with techniques like Retrieval-Augmented Generation (RAG), LLMs can enhance the quality and relevance of the insights they generate, assisting mobile app developers in expediting the responses to the users' feedback to maintain their competitiveness. Thus, the application of LLMs to competitor user review analysis represents a promising approach to automating and improving the process of feature enhancement in mobile apps, ultimately supporting the development of more user-centric and competitive applications.

3 LLM-Cure

LLM-Cure is designed to identify user complaints from user reviews and provide suggestions for developers to enhance features that require the developer's attention. More specifically, *LLM-Cure* operates in two distinct phases: (1) *Scalable Feature Extraction and Assignment* that focuses on identifying and assigning features from user reviews of competing apps and (2) *Suggestion Generation with Competitor Reviews* that leverages the extracted features to identify user complaints from a target app and generate suggestions for feature enhancement leveraging the competitor reviews. By incorporating user feedback from competitor reviews, *LLM-Cure* helps developers address user complaints with a competitive edge. Figure 1 provides an overview of the approach.

```
Below are user reviews, each separated by newline escape. Provide the
list of the top 14 distinct high-level features presented in all the
reviews with a brief meaning of each feature.

High-level features refer to the main functionalities (e.g.,
{functionality 1} or{functionality 2} and the main characteristics (e.g.,
{characteristic 1} and {characteristic 2}) of an app.
For example, the two fine-grained features, {fine-grained feature 1} and
{fine-grained feature 2} belong to the same high-level feature, {high-
level feature}

Below is an example of user reviews with the corresponding high-level
feature.
Review: {review sample 1}
1. {feature}: {feature description}

{User reviews}
```

Fig. 2. Template of *LLM-Cure*'s prompt for feature extraction

3.1 Scalable Feature Extraction and Assignment

Step 1: Extracting Features with Batching and Matching. This step focuses on identifying the top features that can be summarized by LLMs from an extensive collection of user reviews. However, LLMs have limitations on the amount of context they can process at once. To address this challenge and make our approach scalable, we introduce the so-called *batch-and-match* approach that incrementally extracts the top k features from a large corpus of user reviews. Our approach consists of three processes:

① *Batching reviews and extracting features.* Batching reviews involves dividing a large volume of reviews into manageable batches for incremental processing and feature extraction using LLMs. First, we shuffle the entire collection of user reviews of a group of competing apps to ensure randomness. Let $R = \{r_1, r_2, ..., r_m\}$ represent the shuffled user reviews where m is the total number of reviews. To efficiently process the large volume of reviews, we split the shuffled reviews R into batches of a predefined size s (e.g., 1,000 reviews) that fit within the LLM context size. We denote the batches as $B = \{B_1, B_2, ..., B_n\}$, where n is the total number of batches. Each batch B_i corresponds to a set of individual reviews $R_i = \{r_i, r_{i+1}, ..., r_{i+s}\}$, with each r_i being a user review in batch B_i. We process each batch B_i sequentially using the LLM. For each batch B_i, we prompt the LLM to extract the top k features, denoted as F_i, from the set of reviews R_i. The value of k is a hyper-parameter that can be tuned to optimize the performance of *LLM-Cure* depending on the selected dataset. The *Feature extraction* prompt, illustrated in Figure 3, identifies the top k features for a specific app category. It is structured to encapsulate the task description, define features, include a one-shot example, and present the list of user reviews.

② *Matching and merging similar features.* Since we process thousands of reviews in batches due to limitations on the LLM's context handling, the extracted features might sometimes use different

```
Below are user reviews, each separated by newline escape. Provide the
list of the top 14 distinct high-level features presented in all the
reviews with a brief meaning of each feature.

High-level features refer to the main functionalities (e.g., "Blackout
broadcasting" or "Playoff coverage") and the main characteristics (e.g.,
"UI appearance" and "App stability") of an app.
For example, the two fine-grained features, "High provider prices" and
"Cable paid options," belong to the same high-level feature,
"Subscription service."

Below is an example of user reviews with the corresponding high-level
feature.
Review: "The game I want to watch is blacked out."
1. Blackout Broadcasting: feature represents the broadcasting
restrictions imposed on some games.

{User reviews}
```

Fig. 3. Example of *LLM-Cure*'s feature extraction prompt for the Sports News category

wording. For instance, in one batch, the LLM might identify *"Advertisements"* as a feature, while another batch might highlight *"In-app Advertisements"*. Both features represent the same functionality extracted from the user reviews. To ensure a non-redundant feature identification, we address this challenge by incrementally combining similar features after processing each batch B_i. F_i represents the features extracted from batch B_i. We define M_i as the set of matched and merged features until batch B_i. The merging process starts with the top K_1 features extracted from the first batch B_1. For the first batch, $M_1 = F_1$. For subsequent batches (B_i, $i > 1$), we compare features in F_i with M_{i-1} features already identified from the previous batch B_{i-1} and match then merge the similar features. Word embeddings and cosine similarity are employed to achieve the merging. Word embeddings represent the features in a high-dimensional vector space, capturing their semantic meaning [8]. Cosine similarity, a metric for measuring similarity between vectors, is then calculated between the embedding vectors of features F_i and M_{i-1} from different batches. Features exceeding a predefined similarity threshold tr in cosine similarity are considered highly similar and subsequently matched and merged. The similarity threshold tr is a hyper-parameter. Therefore, we experiment with thresholds ranging from 0.7 to 0.85 on a validation set and choose the value that leads to the highest precision. This incremental process continues with each new batch B_i, merging similar features from F_i with the existing merged set M_{i-1} leading to a unique set of features M_i.

③ *Verifying convergence and stabilizing features.* The challenge in this step is to determine when the incremental *batch-and-match* process has sufficiently captured the top k features, avoiding unnecessary iterations that would consume additional processing time and computational resources to process the entire volume of user reviews. We address this by defining a *convergence threshold* based on the stability of the top k features over a specified number of consecutive iterations N. For instance, the system starts with the initial merged set (M_1). The batch processing continues until the merged set M_j where the merged sets remain unchanged across the last N iterations (from M_j up to M_{j-N}). For example, assuming the *convergence threshold* is set to 3. *LLM-Cure* checks if the

top k features identified have remained stable for the last 3 batches. This stability indicates that we have captured the dominant features in the reviews R, and further processing would probably not yield new features. The *convergence threshold N* is a hyper-parameter. Therefore, we experiment with thresholds equal to 3, 5, and 7 on a validation set. Following this convergence step, we obtain the final set of top k features extracted from the review batches.

Step 2: Assigning Features to Reviews. The prior research on the dataset [7] used in our approach demonstrates that only 8.6% of the user reviews contain multiple features and that multi-labeling does not lead to a significant impact on the results. We leverage prior findings to task a language model to assign one feature to the reviews by constructing a *Feature assignment* prompt. The Feature extraction prompt, illustrated in 4, is designed to guide the LLMs in identifying the top k features for a specific app category. It is structured to include the following elements: a description of the task, a definition of the features to be extracted, a one-shot example that demonstrates how the features should be identified, and a list of user reviews that provide the input for analysis. It begins with a clear task description that explains the objective which identifies key features from user reviews. Next, it defines the specific features relevant to the app category being analyzed. To guide the LLM's response, a one-shot example is provided, demonstrating how the model should process the reviews and assign the features. The list of user reviews is then presented for analysis. This design is flexible and can be adapted to different app categories. The task description, feature definitions, and examples can be tailored based on the specific app and the types of features to be identified. Consequently, the template in 4 can be adapted for different app categories by modifying the prompt content accordingly. 5 presents a sample customized prompt for the Sports News category. As a result, each user review r_i is associated with one designated feature, building the groundwork for targeted analysis and feature enhancement suggestions generation in the second phase.

3.2 Suggestion Generation with Competitor Reviews

Prior research [23, 58] shows that negative reviews, those with 1 or 2-star ratings, are particularly interesting to developers as they often contain valuable insights regarding feature complaints and areas for enhancement. Conversely, positive reviews, typically rated 4 or 5 stars, offer detailed descriptions of features and positive user experience [38]. These positive reviews are valuable resources as they often showcase successful implementations of similar features and offer potential solutions to address user complaints. Additionally, prior research [4, 49] indicates that 3-star ratings are typically viewed as neutral, or often encompass both praise and criticism of the app features.

In Phase 2, we leverage the user feedback summarized from competitors' positive reviews to provide suggestions for the target apps to improve the features associated with negative reviews. Our proposed method uses an RAG approach to dynamically construct prompts for the LLM that are augmented with relevant positive user reviews from competing apps. By analyzing these positive reviews, the LLM can identify successful implementations and suggest potential solutions to user complaints within the target application. *LLM-Cure* generates suggestions based on the following five distinct steps.

Step 1: Curating Popular Underperforming Features. An underperforming feature is defined as one that has the largest percentage of negative reviews. Identifying underperforming features is crucial for developers to prioritize areas for improvement and focus on features with the highest percentage of negative reviews to address user dissatisfaction better. We calculate the *Underperforming Feature Score (UFS)* for each feature by determining the percentage of negative reviews

```
Assign only one high-level feature from the list below to each user review.
Assign the category "Other" if you can't find a corresponding category from
the list. Never assign a feature that is not from the list.

User reviews :
{user reviews}

List of 14 high-level features and their meaning :

1.{feature 1: feature description 1}
2.{feature 2: feature description 2}
...
3.{feature 13: feature description 13}
4.{feature 2: feature description 14}


Review 1: {review 1 text}
Output: Review 1,{review 1 text},{Assigned feature}
Review 2: {review 2 text}
Output: Review 2,{review 2 text},{Assigned feature}
Review 3: {review 3 text}
Output: Review 3,{review 3 text},{Assigned feature}
Review 4: {review 4 text}
Output: Review 4,{review 4 text},{Assigned feature}
Review 5: {review 5 text}
Output: Review 5,{review 5 text},{Assigned feature}

Provide the output as CSV content: Review#,"Review text",Assigned Feature.
Do not provide any explanations or notes. Do not include the word "feature".
```

Fig. 4. Template of *LLM-Cure*'s prompt for feature assignment

associated with it. The formula for *UFS* for a particular feature F is:

$$\text{UFS}_F = \frac{\text{Number of Negative Reviews}_F}{\sum_{i=1}^{k} \text{Number of Negative Reviews}_i} \times 100 \tag{1}$$

Where the Number of Negative Reviews$_F$ denotes the number of negative reviews associated with feature F, and $\sum_{i=1}^{k}$ Number of Negative Reviews$_i$ denotes the total number of negative reviews across all k features. Sorting features in descending order of their UFS prioritizes those with the highest percentage of negative reviews. This approach highlights popular features, which are frequently mentioned in user reviews and are more likely to be experienced by a large portion of the user base, allowing developers to focus on those most associated with user dissatisfaction[27].

Step 2: Segregating Positive and Negative Reviews. In this step, for a selected underperforming feature F, we select the negative reviews rated 1 and 2 stars of the target app. Concurrently, we also curate positive reviews rated 4 or 5 stars of the same feature F from competitor apps. We exclude reviews associated with a 3-star rating.

```
Assign only one high-level feature from the list below to each user review.
Assign the category "Other" if you can't find a corresponding category from the
list. Never assign a feature that is not from the list.

User reviews:
{user_reviews}

List of 14 high-level features and their meaning:
1. Device Compatibility: The app's ability to run on various devices and
operating systems.
2. Live streaming: The app's ability to live streaming sports events through the
app.
3. Notifications: The app's ability to send alerts and notifications to users
about games, scores, and news.
4. User interface: The app's design and layout including ease of navigation.
5. Stability and performance: The app's performance, including general quality,
crashes, and freezes.
6. Ads: The presence and frequency of advertisements in the app.
7. Customer support: The app's quality and responsiveness of the customer
support.
8. Score updates: The app's ability to provide real-time scores and updates for
games.
9. Video quality: The app's resolution and overall quality of the video streams.
10. Customization: The app's ability to customize the settings, such as selecting
favorite teams or sports.
11. Subscription service: The app's requirement for a paid subscription to access
certain content or features.
12. Chromecast support: The app's ability to cast content to a TV using
Chromecast.
13. Integration with providers: The app's ability to integrate with other
services, such as social media or cable providers.
14. Sports coverage: The app's variety of types of sports and leagues and
coverage of playoffs and blackouts.

Examples:
Review 1: "I love that Hockey Playoff Nirvana available to be watched."
Output: Review 1, "I love that Hockey Playoff Nirvana available to be watched",
Sports coverage
Review 2: "The game doesn't play on my tablet."
Output: Review 2, "The game doesn't play on my tablet.", Compatibility
Review 3: "A great app but there are far too many notifications I've tried to
turn them off but they're still coming."
Output: Review 3, "A great app but there are far too many notifications I've
tried to turn them off but they're still coming.", Notifications
Review 4: "Game cut out to a black screen during the Stanley Cup playoffs"
Output: Review 4, "Game cut out to a black screen during the Stanley Cup
playoffs", Sports coverage
Review 5: "Excellent way to get access to shows when paying high provider prices"
Output: Review 5, "Excellent way to get access to shows when paying high provider
prices", Subscription service

Provide the output as CSV content: Review#, "Review text", Assigned Feature. Do
not provide any explanations or notes. Do not include the word "feature".
```

Fig. 5. Example of *LLM-Cure*(Mistral)'s prompt for feature assignment for the Sports News Category

Step 3: Prioritizing Complaint-Rich Negative Reviews. Not all reviews contain the same amount of details. Some might express generic dissatisfaction, while others delve deeper and provide specific details about the issues encountered with the feature. In this step, we aim to guide developers towards the most informative negative reviews, i.e., complaint-rich, for a specific underperforming feature F. To accomplish this, we implement a ranking mechanism on the negative reviews associated with the target app, utilizing the TF-IDF (term frequency-inverse document frequency) score [46]. TF-IDF analyzes the importance of words within a document (in this case, a user review) relative to their occurrence across the entire dataset of reviews. For a specific feature F of a target app, we use TF-IDF to calculate a score for each negative review. We sum up the TF-IDF scores of all words in a review to get a final TF-IDF score for the entire review. Higher TF-IDF scores indicate that the user review contains terms that are both frequent in the review and relatively unique across the entire dataset, suggesting detailed and specific feedback. This score reflects the review's richness in terms of feature-specific complaints. We select the top n negative reviews based on the calculated TF-IDF scores. These selected reviews $R = \{r_1, r_2, ..., r_k\}$ guide developers, directing their attention towards negative feedback rich in informative content regarding feature complaints specific to the target app.

Step 4: Identifying Candidate Reviews for Relevant Solutions. This step aims to identify potential recommendations from competitors' user reviews that might address the complaint in a specific negative review. This step consists of four processes:

① *Selecting a negative review.* We start by picking one of the top n negative reviews (e.g., r_1) identified in Step 3.
② *Selecting candidate reviews from competitors.* We look at positive (4 and 5-star) reviews P_F for the same feature F selected in Step 2. To mimic a practical environment where developers might only have access to historical data, we further filter out the candidate positive reviews to include only those with a post date equal to or before the selected target complaint review r_1. These reviews represent the candidates for finding suggestions for improvement.
③ *Creating vector embeddings.* We employ an LLM-based word embedding technique to convert the reviews into vector representations to compare them based on their semantic meaning. Using the same embedding model, we generate the vector representations for the selected negative review r_1 and all positive reviews P_F. Let V_{r_1} represent the vector representation of the selected negative review r_1, and V_{p_i} denote the vector representation of a positive review p_i.
④ *Finding similar reviews.* Since all reviews are represented in the same vector space, we leverage cosine similarity to compare the vectors. The cosine similarity $\text{sim}(V_{r_1}, V_{p_i})$ between the vector representations of the negative review and each positive review is calculated as:

$$\text{sim}(V_{r_1}, V_{p_i}) = \frac{V_{r_1} \cdot V_{p_i}}{\|V_{r_1}\| \cdot \|V_{p_i}\|} \tag{2}$$

We rank the positive reviews based on their similarity, and we identify a sample P of positive reviews that exhibit the highest similarity to the negative review. These selected positive reviews represent instances where users discuss similar feature characteristics but in a positive context.

Step 5: Prompting Suggestions Using RAG. This step revolves around constructing the prompt for instructing the LLM to generate the relevant suggestions. Specifically, we design RAG-based prompts by leveraging the positive reviews identified in Step 4 to provide contextual guidance to the LLM regarding where to draw suggestions. Then, we instruct the LLM to suggest top N

```
You are provided with a user review containing a complain about the {feature}
feature of a {app_category} mobile applications.

Using the below list of positive user reviews about the same {feature} feature
found in competitor apps user reviews, suggest 3 enhancements to address the
highlighted user complain.

User review containing complain about the {feature} feature:
{user_review_complaint}

List of positive user reviews from competitive apps about the {feature} feature:
{candidates_user_reviews}
```

Fig. 6. Template of *LLM-Cure*'s prompt for featue improvement suggestions

unique and constructive recommendations to enhance the feature F, discussed in review r_1, using the provided sample of positive reviews identified in Step 4. Figure 6 illustrates the template for the *Improvement Suggestions* prompt. This RAG-based prompt provides developers with suggestions derived from positive user experiences, facilitating targeted improvements to address user concerns effectively.

3.3 Implementation of LLM-Cure

LLM Choice and Application. In our methodology, we experiment with two different LLMs: *Mixtral-8x7B-Instruct-v0.14* and *GPT-4o mini*[43]. The *Mixtral-8x7B-Instruct-v0.14* model, a high-performing Sparse Mixture of Experts model, was chosen for its balance of cost and performance. The model consists of multiple expert networks, but only a subset is activated during each inference, making it computationally efficient. It has been demonstrated that *Mixtral-8x7B-Instruct-v0.14* surpasses open source models, including Llama 2 70B5 while achieving 6x faster inference, and it matches GPT-3.56 performance on standard tasks [1]. The *Mixtral-8x7B-Instruct-v0.14* model is an open-source model, making it accessible to researchers and developers for replication. The model is installed on a GPU with CUDA and cuDNN for optimized performance, and we use Python scripts to load the model from the Hugging Face Hub[20]. The system requires a GPU with at least 16 GB of memory for efficient inference, and we ensure the inference speed is optimized through the use of Python scripts that interact with the model. The necessary dependencies for this setup include the torch (PyTorch), transformers (Hugging Face Transformers library), cuda (for GPU support), and huggingface_hub, all of which are installed within a Conda environment to guarantee smooth execution.

For *GPT-4o mini*, a closed-source variant of GPT-4, we access it using the API provided by OpenAI[4]. To interact with the *GPT-4o mini*, we obtain an API key from O which is required for authentication. We use the official Python API client to send requests to the model and retrieve responses. To ensure deterministic output, we set the temperature parameter to zero. This approach, being cloud-based, allows us to scale the analysis without the need for specialized hardware. The interaction with the *GPT-4o mini* model is authenticated through the API key.

Result Parsing. After receiving the output from the LLMs, we parse the results using a customized Python script. Since our prompts specify the required output format, the LLM responses are

[4]https://platform.openai.com/docs/overview

structured accordingly. For example, in the Feature Extraction Prompt, the output consists of a list of features with their associated brief description. We parse this response to generate a .CSV file with two columns: "Feature" and "Feature Description." Similarly, for the Feature Assignment Prompt, we parse the LLM's output to create a .CSV file that contains each user review along with the assigned feature. This structured approach ensures that the data generated from the LLM outputs is efficiently organized and ready for further analysis or integration into the app improvement process.

Embedding Model Choice. To ensure consistency within our approach, *LLM-Cure* employs the *mistral-embed*[5] word embedding model from Mistral AI during the process that requires word embedding. Specifically, we used it in the *Matching Similar Features* process of phase 1 and in the *Identifying Candidate Reviews for Relevant Solutions* step of phase 2. We leveraged the Mistral API to retrieve text embeddings efficiently.

Text Preprocessing. Prior to feeding text inputs into the *mistral-embed* model, we conducted standard text normalization processes adopted in previous work [5, 6] to enhance the quality of the input data by applying tokenization, removal of stop words, stemming, and spell-checking. We employed the SpellChecker[6] along with the nltk[7] libraries in Python for these preprocessing steps.

4 Experimental Design

We aim to evaluate the ability of *LLM-Cure* to (1) accurately assign features to user reviews and (2) provide the developers with suggestions to improve the features of their apps given a specific complaint. In this section, we describe the experimental setup and discuss the results of our investigation.

Table 1. Descriptive statistics of 7 competing apps categories

App category	Number of apps	Number of reviews
Free call	10	291,034
Weather	10	276,941
SMS	10	264,926
Bible	10	64,417
Music player	10	60,685
Sports news	10	57,582
Cooking recipe	10	41,154
Total	70	1,056,739

4.1 Dataset

We employ the same dataset utilized by a previous work [7]. The original dataset is collected following a systematic approach for selecting and categorizing competing apps. The dataset is built by initially selecting the top 2,000 free-to-download popular apps from the Google Play Store, based on the App Annie report [3]. To ensure diversity and avoid bias toward specific app categories, the study focuses on 29 keywords representing main app features across 17 different Google categories.

[5]https://docs.mistral.ai/api/
[6]https://pypi.org/project/pyspellchecker/
[7]https://pypi.org/project/nltk/

Apps are grouped into 20 distinct categories (e.g., Navigation, Weather, Browser, FreeCall and Dating) based on these features. Each category includes a sufficient number of competing apps (e.g., 8 to 10 competing apps) to facilitate competitor user review analysis. This approach aims to capture a broad range of app functionalities and ensure that each group represents closely related competing apps. Each app category contains 9 or 10 competing apps. User reviews for the 20 apps are collected, resulting in a total of 14,043,999 reviews. To evaluate our approach against the baselines, we use the same five categories used by the baselines [7] to evaluate the precision, namely Weather, SMS, Bible, Music Player, and Sports news. Similar to previous work, we use the Free call and Cooking recipe categories for hyper-parameter tuning. Specifically, for each category, we utilize the same statistically representative sample of reviews, i.e., 96 user reviews per category, resulting in a total of 672 ground truth user reviews to evaluate our approach. Table 1 summarizes the descriptive statistics of the 70 selected competing app categories. The user reviews of each app are also available, and each user review records the title of the user review, detailed comment, user rating and the posting date. Additionally, we have access to the release notes of the apps which allows us to gain information about the features and updates introduced in each app version.

To evaluate our approach against the baselines, we use the same five categories used by the baselines [7] to evaluate the precision, namely *Weather, SMS, Bible, Music Player*, and *Sports news*. Similar to previous work, we use the *Free call* and *Cooking recipe* categories for hyper-parameter tuning. Specifically, for each category, we utilize the same statistically representative sample of reviews, i.e., 96 user reviews per category, resulting in a total of 672 ground truth user reviews to evaluate our approach. Table 1 summarizes the descriptive statistics of the 70 selected competing app categories. The user reviews of each app are also available, and each user review records the title of the user review, detailed comment, user rating and the posting date. Additionally, we have access to the release notes of the apps which allows us to gain information about the features and updates introduced in each app version.

4.2 Research Questions

4.2.1 RQ1: How effective can LLMs be in extracting features from user reviews?

In our work, *LLM-Cure* suggests enhancements to developers to improve their app features by identifying features from user reviews. Therefore, we want to evaluate the capabilities of the LLM in automatically extracting meaningful features from user reviews to understand the feasibility and potential of *LLM-Cure* for real-world applications.

Evaluation Metrics. We assess the performance of *LLM-Cure* by comparing predicted features by the LLM against the ground truth. We employ three key metrics: 1) True Positives (TP) representing the correctly predicted features, 2) False Positives (FP) representing the number of falsely predicted features, and 3) False Negatives (FN) representing features present in reviews but not predicted by *LLM-Cure*. We adopt the precision, recall and F1-score as evaluation metrics, and we calculate them as follows:

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F_1\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{5}$$

To evaluate the performance metrics of feature extraction, the first and second authors, as two independent annotators, manually label 672 testing reviews. The Cohen's Kappa agreement score [13] is computed on the annotated testing reviews, yielding a high score of 0.82, indicative of a high level of agreement.

Experimental Setup. *LLM-Cure* has three hyper-parameters: k the number of the features, the *similarity threshold* and the *convergence threshold*. Previous work that uses the dataset demonstrates that 14 leads to the best results when set as the number of features. Therefore, we set k as 14, aligning with previous work on this dataset [7]. To set the *similarity threshold* and the *convergence threshold*, we conduct experiments on the two validation sets, i.e., *"Recipe cooking"* and *"Free Call"* app categories. The results indicate that a *similarity threshold* of 0.75 coupled with a *convergence threshold* of 5 yields the highest precision. Therefore, we adopt these hyper-parameter values for the testing set. In addition, to prevent exceeding the limited context size of the LLM, we select a batch size of 1,000 reviews. This batch size is determined based on the varying lengths of individual reviews. Through experimentation, we have found that 1,000 reviews is optimal, as it ensures that the total token count of each batch does not exceed the context window of the selected models.

Baselines. To assess the efficacy of *LLM-Cure* in automatically identifying and assigning features to user reviews, we compare its performance against existing baselines FeatCompare[7] and Attention-based Aspect Extraction (ABAE) [28], using the ground truth of 480 labeled reviews. In addition, we introduce a baseline called *LLM-Basic*. Similar to *LLM-Cure*, *LLM-Basic* prompts the LLM to extract features from user reviews. However, unlike *LLM-Cure*, *LLM-Basic* does not incorporate the incremental batching process (batch-and-match) that enhances feature extraction in *LLM-Cure*. Instead, we select a statistical sample of reviews from the total set, ensuring it fits within the context window of the LLM (i.e., 1,000 reviews), and use the same prompt as in *LLM-Cure*. The purpose of this baseline is to evaluate the added value of the incremental batching process in improving feature extraction and overall performance. We implement *LLM-Basic* using two different LLMs: LLM-Basic(GPT), which employs *ChatGPT-4o mini*, and LLM-Basic(Mistral), which uses the *Mixtral-8x7B-Instruct-v0.14*.

Prompt Construction. To construct the prompts, we adhere to the template provided by Mistral for our selected model[8]. *LLM-Cure* leverages two distinct prompts for Phase 1: the *Feature extraction* prompt and *Feature assignment* prompt. The *Feature extraction* prompt is illustrated in Figure 3 and the *Feature assignment* in Figure 5. The prompts are also available in our research artifact.

Results. *LLM-Cure* is capable of identifying and assigning features with high F1-score, recall and precision. As shown in Table 5, across the five testing app groups, LLM-Cure exhibits F1-scores ranging from 80% to 91%, with precision between 81% and 92% and recall between 80% and 90% for LLM-Cure(Mistral). Similarly, LLM-Cure (ChatGPT) achieves an F1-score ranging between 84% and 90%, with precision between 83% and 90% and recall between 83% and 90%. The results of the extracted features for LLM-Cure (GPT) are similar to those of LLM-Cure (Mistral), and both sets of results are included in the replication package. Table 2 and Table 3 show the fourteen features extracted for the five testing apps categories using the Mistral LLM. These findings underscore the capability of LLMs to extract features from user reviews without requiring manual annotation.

[8]https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

Table 2. Top fourteen features extracted from the user reviews of three sample app categories (Part 1)

Feature	Description
Weather	
Accuracy	App's ability to provide accurate weather forecasts
Radar	App's radar and map and visualization features
Weather Forecast	App's hourly and daily forecasts
Additional Features	App's additional features, such as pollen counts and UV index
Notifications	App's ability to send notifications for weather alerts
Customization	App's ability to be customized, i.e., adding multiple locations
Battery Usage	App's impact on the device battery life
Customer Support	Users' experiences with the app's customer support
Ease of Use	App's ease of use including widgets and how to navigate
Ads	App's use of ads, including how intrusive they are
Device Compatibility	App's compatibility with different devices and systems
Design and Layout	App's design and layout
Performance	App's stability, including how often it crashes or freezes
Updates	App's frequency and quality of updates
SMS	
Group Messaging	App's ability to send messages to multiple recipients at once
Dual SIM Support	App's ability to support dual SIM devices
Account Authentication	App's ability to identify and authenticate a user's account
Customization	Ability to customize the app's appearance and/or functionality
Spam Identification	App's ability to identify and filter out spam messages
MMS Support	App's ability to send and receive multimedia messages
Integration with Other Apps	App's ability to integrate with other apps
In-app Ads	Presence of advertisements within the app
Notifications	App's ability to send notifications for incoming messages
Backup and Restore	App's ability to back up and restore messages and settings
User Interface (UI)	App's design and layout
Call Blocking	App's ability to block unwanted calls
Caller Identification	App's ability to identify the caller's name and/or location
Privacy	App's ability to protect the user's privacy
Sport News	
Device Compatibility	App's ability to run on various devices and systems
Live Streaming	App's ability to live stream sports events
Score Updates	App's ability to provide real-time scores and updates for games
Sports Coverage	App's variety of types of sports and coverage
Notifications	App's ability to send alerts and notifications
User Interface	App's design and layout including ease of navigation
Chromecast Support	App's ability to cast content to a TV using Chromecast
Performance	App's performance, including general quality and crashes
Ads	The presence and frequency of advertisements in the app
Customer Support	App's quality and responsiveness of the customer support
Video Quality	App's resolution and overall quality of the video streams
Customization	App's ability to customize the settings
Subscription Service	App's requirement for a paid subscription
Providers Integration	App's ability to integrate with other services (cable providers)

Table 3. Top fourteen features extracted from the user reviews of three sample app categories (Part 2)

Feature	Description
Music Player	
Playlist Management	App's features for creating, editing, and organizing playlists
Lyrics Integration	App's ability to display lyrics in real-time and for offline use
Equalizer & Sound Adjustment	App's options for sound customization
Ad-Free Experience	App's option to remove ads
Search	App's ability to allow users to search for a specific song
Volume Leveler	App's ability to adjust songs' volume to a common level
Shuffle	App's ability to shuffle songs in a playlist or library
Sleep Timer	App's option to automatically stop playback
Sound Quality	App's overall sound quality, including volume and bass
User Interface	App's design and layout (e.g., theme customization)
Notifications	App's ability to send notifications based on user's preference
Offline Mode	App's ability to download songs for offline playback
Chromecast/AirPlay Support	App's wireless streaming with Chromecast or AirPlay
Download	App's ability to download music for offline listening
Bible	
Bible Versions	The ability to switch between different translations of the Bible
Daily Verses	Daily verse for meditation and reflection
Reading Plans	Guided plans for reading the Bible over a set period
Social Features	The ability to share verses or reflections with users
Highlighting & Bookmarking	The ability to mark specific verses for future reference
Search Function	The ability to search for specific verses or topics in the Bible
Audio Feature	The option to listen to the Bible being read aloud
Customization	App's ability to personalize with different themes and fonts
Offline Access	Offline Access and download portions of the Bible
Devotionals	Pre-written devotionals on various topics
User Experience	App's layout design and user experience
In-App Purchases	App's option to buy extra features or content
Notifications	App's ability to send reading reminders
Comparison Feature	App's ability for comparison of different bible translations

LLM-Cure significantly outperforms LLM-Basic, FeatCompare and ABAE baselines across the testing apps. Table 5 shows that on average, *LLM-Cure* achieves a 7% improvement in F1-score, a 9% improvement in recall and a 4% in precision as compared to the baselines. To quantitatively assess these differences, we conducted paired t-tests. A paired t-test [41] is a statistical method used to determine whether there is a significant difference between the means of two related groups. In our case, these groups are the performance metrics of LLM-Cure and FeatCompare, the best performing baseline. Our findings indicate that LLM-Cure significantly outperforms FeatCompare in both F1-score and precision. Specifically, the paired t-test for F1-score yielded a t-statistic of 3.723 with a p-value of 0.02, confirming a statistically significant difference. Similarly, the paired t-test for precision resulted in a t-statistic of 4.784 and a p-value of 0.009, further underscoring LLM-Cure's superior performance.

The *batch-and-match* process of *LLM-Cure* improves the performance of feature extraction. *LLM-Basic*, which only processes a single batch of reviews, achieves lower performance compared to *LLM-Cure*. On average, *LLM-Cure* improves the F1 score by 9% for *LLM-Cure(Mistral)* and 4% for *LLM-Cure(GPT)*. These results highlight the benefit of *LLM-Cure*'s incremental processing, batch-and-match, and its ability to extract features more effectively and with higher performance (e.g., F1-score). Instead of processing the entire set of user reviews, *LLM-Cure* processes only a fraction of the total reviews. Table 4 illustrates the percentage of user reviews needed by *LLM-Cure* to achieve convergence and extract the features. *LLM-Cure* outperforms all baseline methods while processing only between 3% and 30% of user reviews when using Mistral model and between 4% and 31% for GPT, achieving feature saturation without the need for processing the entire dataset.

***LLM-Cure* performs consistently across different sentiment categories.** To further investigate whether *LLM-Cure* classifies positive versus negative reviews with different precision, we conduct an analysis based on the sentiment categories. We find that positive and negative reviews present similar results across categories, with at most 3% of differences. The average precision for positive reviews across these categories is 85.69%, while for negative reviews, it is 85.78%. These findings indicate that there is no significant difference in classification F1-score between positive and negative reviews, demonstrating that our approach is not sensitive to the sentiment of reviews.

Table 4. Number of batches and percentage of reviews required to extract features using LLM-Cure for two models. 'Num.' denotes the number of batches and 'Per.' denotes the percentage of reviews. LLM-Cure (Mistral) refers to LLM-Cure implemented with the Mixtral-8x7B-Instruct-v0.14 model, while LLM-Cure (GPT) refers to LLM-Cure implemented with GPT-4o mini.

App category	**LLM-Cure (Mistral)**		**LLM-Cure (GPT)**	
Num. of batches	Per. of reviews	Num. of batches	Per. of reviews	
Weather	7	3%	12	4%
Bible	4	6%	5	8%
SMS	24	9%	10	4%
Sport News	8	14%	18	31%
Music player	19	31%	8	13%

Table 5. Performance comparison of *LLM-Cure* and baselines on five testing app groups in feature assignment. 'P' denotes Precision, 'R' denotes Recall, and 'F_1' denotes F1-score.

App	**LLM-Cure** (Mistral)			**LLM-Basic** (Mistral)			**LLM-Cure** (GPT)			**LLM-Basic** (GPT)			**FeatCompare**			**ABAE**		
Category	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Weather	**92**	**90**	**91**	80	77	79	85	83	84	82	78	80	81	74	78	64	64	67
Sports News	81	80	80	67	65	66	**90**	**90**	**90**	82	82	82	82	75	78	71	65	68
Bible	**83**	**83**	**83**	78	75	77	83	83	83	83	82	82	81	77	79	72	69	70
SMS	**86**	83	**85**	77	76	77	85	**85**	85	82	81	82	80	74	77	67	62	64
Music Player	86	86	86	82	82	82	**90**	**90**	**90**	86	87	87	79	75	77	70	66	68
Average	86	84	85	77	75	76	**87**	**86**	**86**	83	82	82	82	75	78	68	65	68

Summary of RQ 1

LLM-Cure achieves promising F1-scores ranging from 80% to 91% across the five test sets and 83% and 90% for *LLM-Cure(Mistral)* and *LLM-Cure(GPT)* respectively. These results demonstrate *LLM-Cure*'s effectiveness in analyzing user reviews without manual data labeling. The obtained F1-scores surpass the performance of the baselines by an average of 7% and 8% for *LLM-Cure(Mistral)* and *LLM-Cure(GPT)* respectively. The batch-and-match process enables *LLM-Cure* to achieve a high F1-score with a substantial reduction in the required user review data, ranging from 3% to 31% for *LLM-Cure(Mistral)* and 4% to 31% for *LLM-Cure (GPT)* across the five app categories.'

4.2.2 RQ2: Can LLMs leverage categorized user reviews to generate suggestions for feature improvements?

RQ2 investigates whether *LLM-Cure* can leverage categorized user reviews of competitor apps to generate suggestions for feature improvements. Analyzing competitors' positive user reviews allows developers to identify successful features and user preferences across the market, ensuring their app remains competitive and relevant. By incorporating these insights from competitor reviews, *LLM-Cure* empowers developers to make data-driven decisions about feature enhancement, prioritize user needs, and ultimately create a more competitive app.

Suggestions validation. To evaluate the relevance of the suggestions generated by *LLM-Cure*, we employ a retrospective validation approach at the app release level. Our methodology captures a snapshot of the app version corresponding to the selected user review complaint, ensuring that the complaint is tied to a specific app version at the time when the review was posted. After establishing this time-wise snapshot, we assess the relevance of the suggestions by analyzing the release notes of subsequent app versions. Specifically, we track the features introduced in future app releases that correspond to the complaints mentioned in the user reviews, and calculate the *Suggestions Implementation Rate (SIR)*. This process allows us to measure the extent to which the suggestions provided by *LLM-Cure* are addressed in future updates, offering insight into their practical applicability and the effectiveness of the approach over time. We define the *SIR* the number of suggestions by *LLM-Cure* matched in the release notes divided by the total number of suggestions provided as follows:

$$\text{SIR} = \frac{\text{Number of Suggestions Matched in Release Notes}}{\text{Total Number of Suggestions by LLM-Cure}} \tag{6}$$

Experimental Setup. Experimental Setup. In RQ1, we utilized two LLMs to demonstrate the effectiveness of our approach. Since both models produced similar results, we proceeded with RQ2 by selecting the *LLM-Cure(Mistral)* model for further evaluation. Mistral model is an open-source model, which enables the reproducibility of our experiments by other researchers as well as developers, and helps avoid the cost associated with closed-source models. We randomly select three categories to evaluate the feature improvement suggestions. From each category, We select the apps with a substantial number of informative release notes to ensure that we have a rich data source for conducting the manual suggestion validation. Specifically, we choose *Handcent Next SMS messenger*[9]

[9]https://play.google.com/store/apps/details?id=com.handcent.app.nextsms

Table 6. Suggestions Implementation Rate (SIR) of *LLM-Cure* Feature Improvement Suggestions on the selected three apps. Avr. denotes Average and Underperf. denotes Underperforming

App category	App name	# of releases	Avr. words per release	Underperf. features	# of neg. reviews	# of pos. reviews	SIR
Weather	Weather & Clock Widget	44	36	Accuracy	336	8,383	8/9 = 89%
				Ease of Use	159	4,313	5/9 = 56%
				Performance	157	1,951	4/9 =44%
SMS	Handcent Next SMS Messenger	140	30	User Interface	316	686	9/9 = 100%
				Notifications	297	359	6/9 = 67%
				MMS Support	229	162	7/9 = 78%
Sports News	FOX Sports: Watch Live	36	23	Performance	27	1,947	8/9 = 89%
				Sports Coverage	26	5,640	4/9 = 44%
				Notifications	18	6,558	8/9 = 89%
Total SIR							**59/81 = 73%**

from the SMS category, *FOX Sports: Watch Live*[10] from the Sports News category, and *Weather & Clock Widget*[11] from the Weather app category. As shown in Table 6, the chosen Weather, SMS, and Sports News apps have 40, 140, and 36 release notes, respectively, with average word counts of 36, 30, and 23 per release. For each app, we apply steps 1 to 5 of *LLM-Cure*'s Phase 2. We focus on the top three underperforming features that require the most attention from developers. For each feature, we identify three target complaints. For each user complaint, we generate suggestions to improve the app features. We obtain a total of 9 suggestions per feature, resulting in 27 suggestions per app. We then calculate *SIR* for each feature.

Results. If *LLM-Cure* had been used, it would have identified 59 out of 81 feature enhancements (i.e., 73%) that were later implemented by developers, demonstrating its potential to align with real-world improvements. Table 6 shows the SIR for each feature across the three apps. The results indicate that some features received higher *SIRs* than others. For example, all the User Interface-related suggestions for *"Handcent Next SMS messenger"* were implemented, while not all Notification-related suggestions for the same app were adopted. This variation can be attributed to different development priorities or challenges associated with certain feature enhancements. Furthermore, the release notes sometimes contained high-level descriptions of updates, such as "Performance improvement," which may not explicitly detail the changes but could reflect the overall enhancement suggested by *LLM-Cure*.

***LLM-Cure* successfully leverages candidate positive user reviews from competitors to provide suggestions for feature improvements related to user complaints.** We illustrate this process through two case studies, randomly selecting underperforming features from the *Weather & Clock Widget* app. For the underperforming *"Ease of Use"* feature, we consider the below user review that highlights an issue where the weather widget fails to update automatically, requiring manual refreshes:

> *"Widget won't update over time it keeps showing same weather and same TIME until I tap on it and open weather app."* ★☆☆☆☆

[10]https://play.google.com/store/apps/details?id=com.foxsports.android
[11]https://play.google.com/store/apps/details?id=com.devexpert.weather

The target review identified two weaknesses in the widget: **(1) unreliable timekeeping** and **(2) an unfriendly design**. *LLM-Cure* tackles these issues by proposing an automatic refresh functionality, an improved time synchronization, and the implementation of a refresh icon. These suggestions directly address the user's frustrations and aim to improve the widget's usability. Specifically *LLM-Cure* proposes the below three suggestions:

LLM-Cure suggestions for the Ease of Use complaint

Automatic Refresh: Implement an automatic refresh feature for the widget that updates the weather and time data periodically, so users don't have to manually tap on the widget to see the current weather and time.

Improved Time Synchronization: Enhance the clock functionality in the widget to ensure that it stays in sync with the device's internal clock, eliminating the need for manual time updates.

Clear Refresh Icon and Options: Provide a clear refresh icon and add options for users to manually refresh the widget, change clock fonts, and customize the weather update intervals to their preference. This will give users more control over the widget's behavior and improve their overall experience.

Subsequently, we cross-referenced these suggestions with the app's release notes and found that developers implemented recommendations provided by *LLM-Cure* in future releases. For instance, in release *6.1.0.1*, the functionalities *"Added option to show forecast every 3 hours on the widget when you select hourly forecast"* and *"Enabled digital font for clock and date"* were introduced, aligning closely with the suggestions aimed at enhancing the time synchronization of the widget.

Following the same approach outlined above, we select the below user complaint belonging to the *Radar* underperforming feature:

"Needs a radar view for everyone's local area. Not having it takes away from the app." ★★☆☆☆

The target review complains about the missing radar view for the local user area. The complaint highlights a shortcoming in the app's radar functionality on two fronts: **1) Radar location**, i.e., the user can't easily see weather patterns in their immediate surroundings and **2) Radar view perspectives**, i.e., the app lacks the flexibility to customize the radar display. *LLM-Cure* addresses these issues by proposing customizable views, GPS integration, and real-time radar updates leveraging relative competitive user reviews. Table 7 illustrates a sample of the candidate reviews.

Specifically, *LLM-Cure* offer the below three suggestions:

LLM-Cure suggestions for the Radar complaint

Customizable Radar Views: Some users have mentioned that they would like more control over the radar view, such as the ability to adjust the time frame or overlay different data layers. Adding customizable radar views could help users get the information they need for their specific location.

Integration with GPS: Several users have praised radar features that integrate with GPS to automatically show their current location on the map. Adding this feature could help users quickly access radar information for their local area, even if they are not familiar with the region.

Real-time Radar Updates: Many users have praised radar features that update in real-time, allowing them to track weather systems as they develop. Implementing real-time radar updates could help address the user's complaint about the app not having a radar view for their local area.

LLM-Cure's suggestions directly target the weaknesses identified in the user complaint, demonstrating its ability to analyze user complaints and propose relevant improvements leveraging the reviews of competing apps. Furthermore, we analyzed the future release notes for the *"Weather & Clock Widget"* app after the user's complaint (*release 5.9.1.3*). The improvements mentioned in future releases aligned with the LLM's suggestions. For instance, *release 6.0.0.1* introduced a "weather radar service screen," directly addressing the lack of a local radar view. Additionally, *release 6.0.1.2* offered the *"option to set radar default layer"*, which aligns with the suggestion for customizable views.

Table 7. A sample of five reviews regarding the *Radar* feature from competitors of the *"Weather & Clock Widget"* app

Competitors' user reviews
"I like that the radar is on the first page. Lots of times I just want to see where the rain/snow/storm is and how close it is. I don't have to go through a bunch of screens to get there."
"I love that it puts my current location on the radar maps. Great interface."
"Radar update with gps location is very accurate. It helps me in my road trip planning avoiding snow hazards. Loved it. It has variety of radar scans temperature etc."
"I like the live radar. Wish it was a longer time Frame though."
"Exactly what I need. Easy to find radar. Easy to view by location or other areas."

***LLM-Cure's* suggestions often align with functionalities later implemented by the apps documented in the release notes.** Table 8 presents a random sample of *LLM-Cure* suggestions alongside corresponding release notes selected from 81 of the total suggestions, demonstrating the alignment between the feature improvement suggestions and the actual implementations. These results provide encouraging evidence that *LLM-Cure* can effectively analyze competitors' user reviews and generate suggestions for feature improvements.

Table 8. Sample of LLM-Cure's suggestions and corresponding release notes

LLM-Cure's Suggestions	Release Notes
Add a customizable notification sound: Allowing users to set their preferred notification sound can significantly improve their experience	Fixed personalized sound notification
Improve the quality of MMS messages: The app should allow users to send high-quality images and videos through MMS	High quality picture compression when MMS
Reliable MMS delivery: Improve the MMS delivery system to ensure that messages are sent and received reliably	Improved send/receive MMS known issues
Improve international sports coverage: Some users have requested better coverage of international sports, such as rugby and cricket	Expanded golf coverage including schedule, leaderboards, scorecards, tee times, and rankings
Add a dedicated tab for NHRA events: To address the user complaint, the app could add a specific tab for NHRA events, making it easier for users to find and access coverage of these events	Enhanced home screen with all your favorite team scores, news, and live streams in one place

Summary of RQ 2

LLM-Cure successfully leverages candidate positive reviews of competitors to generate feature improvement suggestions for user complaints. *LLM-Cure* achieves a promising average of 73% of Suggestions Implementation Rate (SIR), demonstrating its potential for competitive feature enhancement.

5 Threats to Validity

Threats to construct validity relate to a possible error in the data preparation. In *LLM-Cure*, we adopt a batch-and-match methodology to accommodate a scalable LLM prompting by employing a subset of reviews to extract features. We evaluate the validity of our approach by testing it across various thresholds and validating against ground truth data, ensuring robustness in representation. To reduce any bias that may be introduced by the order of reviews, we shuffle user reviews before splitting them to ensure a representative sample of user reviews.

In Phase 2, i.e., the Suggestion Generation with Competitor Reviews phase, we suggest addressing the most underperforming features, i.e., having the highest number of negative reviews in the target app. We acknowledge that developers may employ different prioritization (e.g., severity-based, effort-based) techniques in the real world. However, our methodology remains valid. Our approach is anchored in the actual reviews rather than the selection approach, thus ensuring the reliability of our results. Developers have the flexibility to prioritize and address complaints from any features, whether they are functional or non-functional.

Threats to internal validity relate to the concerns that might come from the internal methods used in our study. One threat may stem from the selection and design of the prompt templates. To address this potential threat, we explore different prompts. When provided with the same shot examples, we observe that the prompt template does not lead to different classification results. All

the used prompts are available in the replication package.

Threats to external validity concern the ability to generalize the results. One threat concerns the ability to generalize the results. Specifically, the choice of the LLM utilized in our approach, *LLM-Cure*, is a key factor. We acknowledge that each LLM possesses a distinct architecture, potentially leading to variations in results. However, we opt for *Mixtral-8x7B-Instruct-v0.14* (an open-source model) and *GPT-4.0 mini* (a closed-source model) for several reasons. As detailed in Section 3.3, both models achieve similar results across various benchmarks, demonstrating their effectiveness. Mistral is openly available for researchers, fostering the replication of our work and ensuring transparency in the evaluation process, while *GPT-4.0 mini* offers strong performance as a proprietary alternative. Despite these considerations, it's important to recognize that the choice of LLM remains a potential source of variability in our findings. Another threat to external validity pertains to the selection of categories and datasets. However, similar to prior research[7], we mitigate this concern by evaluating the validity of our approach across five distinct app categories sourced from ground truth data. This approach aims to minimize the influence of app selection bias. Additionally, our method is platform-agnostic, offering applicability to any mobile app, provided it contains user reviews. This broad applicability enhances the generalizability of our findings beyond specific categories or datasets.

6 Related Work

Feature Enhancement. Prior approaches have explored automatic feature extraction from user reviews for feature enhancement [16, 26, 36, 61]. For example, Scalabrino et al. [16] introduce CLAP, a web application facilitating mobile app release planning by analyzing user reviews. CLAP categorizes reviews from the target app, prioritizing user concerns to be addressed. Wang et al. [61] present UISMiner, which supports UI-related feature enhancement by mining user review suggestions about UI. Gao et al. [26] propose a method for analyzing user reviews to extract requirements and update app goal models, including feature improvements and additions. However, these contributions focus solely on individual target apps, lacking consideration of competing apps. Liu et al. [36] present an approach considering market trends, guiding developers on feature update strategies by comparing features of similar apps. While Liu et al.'s work includes a competitive analysis aspect, it primarily suggests which features to update rather than offering suggestions for improvement.

Similar to the aforementioned contributions, our aim is feature enhancement. However, our work distinguishes itself in two key aspects: (1) we aim to guide developers towards feature enhancements automatically, irrespective of the feature categories; and (2) we offer a competitive landscape for feature improvement by harnessing the competitors' user reviews and the power of LLMs. This allows us to not only suggest enhancements based on user complaints but also provide insights into how these features compare with those of competing apps.

Mobile Apps Competitor Feature Analysis. Researchers have proposed methods for extracting detailed, fine-grained features from user reviews and comparing them across app competitors. Shah et al. [54] introduce REVSUM, a competitor analysis tool that evaluates sentiment, bug reports, and feature requests between a target app and its competitors. Dalpiaz and Parente [14] introduce RE-SWOT, which constructs a Strength-Weakness-Opportunity-Threat (SWOT) matrix for competitor analysis. Another tool by Shah et al. [51] compares two competing apps based on the selected features and analyzes competitor sentiment. However, relying on fine-grained features for competitor analysis is impractical due to the large number of features that can be extracted [7, 14, 52]. In response, Featcompare [7] proposes an automatic approach to mine high-level features

(i.e., semantic clusters of fine-grained features) from competing apps and compares user sentiment across these features. However, none of these approaches offer developers suggestions for improving features. Wang et al. [60] introduce a UI-focused feature recommender tool for enhancing app competitiveness by recommending missing features based on the analysis of UI pages from similar apps. While Wang et al.'s work recommends missing features from a target app, it is only limited to the UI features.

LLMs for Mobile Apps. De Lima et al. [35] propose a method utilizing LLMs to autonomously identify risk factors from app reviews and prioritize them to anticipate and mitigate risks. Roumeliotis et al. [48] conduct an evaluation study comparing the effectiveness of LLMs like Llama and GPT 3.5 in predicting sentiment analysis related to e-commerce. Similarly, Zhang et al. [66] assess the performance of three open-source LLMs in zero-shot and few-shot settings for predicting sentiment in user reviews. Xu et al. [64] design a prompt instructing ChatGPT to extract aspect-category-opinion-sentiment quadruples from text. Wei et al. [63] propose Mini-BAR, a tool integrating LLMs for zero-shot mining of bilingual user reviews in English and French. Dos Santos et al. [19] analyze accessibility reviews using LLMs. While the above work leverages user reviews, Huang et al. [63] introduce CrashTranslator, which automatically reproduces mobile app crashes from stack traces guided by LLMs to predict the exploration steps for triggering the crash. Liu et al. [37] propose InputBlaster, leveraging LLMs to generate unusual text inputs for mobile app crash detection. In our approach, we harness LLMs to provide suggestions for feature improvement, complementing the aforementioned contributions to support mobile app analysis.

7 Conclusion

In this work, we introduce *LLM-Cure*, a novel LLM-based approach that conducts Competitor User Review Analysis for Feature Enhancement. *LLM-Cure* generates automated suggestions for app feature improvements by leveraging user reviews from competitor apps to enhance user experience and maintain competitiveness. Specifically, *LLM-Cure* leverages LLMs to extract and semantically categorize features from user reviews eliminating overly granular feature sets that hinder prior methods. Then, it curates underperforming features in the target app and suggests improvements by analyzing highly rated similar features in competing apps. Unlike previous methods that rely on explicit comparisons or simple keyword-based matching, *LLM-Cure* refines feature analysis through semantic categorization and generates actionable suggestions, helping developers make informed decisions. We evaluate *LLM-Cure* on 1,056,739 reviews of 70 popular Android apps. *LLM-Cure* achieves high performance in extracting and assigning features to user reviews, outperforming baseline methods significantly by up to 13% in F1-score, up to 16% in recall and up to 11% in precision. *LLM-Cure* achieves a promising suggestions implementation rate of 58 out of 81. By combining user feedback with competitor user review analysis, *LLM-Cure* empowers developers to make informed decisions, fostering a more competitive landscape.

In the future, we aim to explore agent orchestration between LLMs to enhance the feature analysis process and leverage the complementary strengths of different language models. Additionally, we aim to expand *LLM-Cure*'s capabilities by prioritizing suggestions based on factors such as popularity and potential user impact. To achieve this, we propose conducting a developer-involved case study to gain deeper insights into how feature improvement suggestions align with real-world development priorities. This approach will help refine our methodology to better support practical decision-making. Furthermore, to facilitate the practical adoption of *LLM-Cure*, we plan to explore various deployment options. One possibility is to package *LLM-Cure* as a standalone web-based tool with a user-friendly interface, allowing developers to easily upload user reviews and receive feature enhancement suggestions. We also aim to release *LLM-Cure* as an open-source Python

package, enabling practitioners to integrate it into their own analytics pipelines and customize it to fit specific project needs. By making *LLM-Cure* accessible in these ways, we hope to empower developers to leverage competitor user review analysis more effectively in real-world development scenarios.

Data Availability. We provide a replication package including the data and scripts to replicate the analyses at https://github.com/repl-pack/LLM-Cure.

References

[1] Mistral AI. 2024. Mixtral of Experts. https://mistral.ai/news/mixtral-of-experts/

[2] A. AlSubaihin, F. Sarro, S. Black, L. Capra, and M. Harman. 2019. App Store Effects on Software Engineering Practices. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2891715

[3] AppAnnie. 2016. App Annie. https://www.appannie.com/. Last accessed March 2020.

[4] Rahul Aralikatte, Giriprasad Sridhara, Neelamadhav Gantayat, and Senthil Mani. 2018. Fault in your stars: an analysis of Android app reviews. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, COMAD/CODS 2018, Goa, India, January 11-13, 2018*, Sayan Ranu, Niloy Ganguly, Raghu Ramakrishnan, Sunita Sarawagi, and Shourya Roy (Eds.). ACM, 57–66. https://doi.org/10.1145/3152494.3152500

[5] Nimasha Arambepola, Lankeshwara Munasinghe, and Nalin Warnajith. 2024. Factors Influencing Mobile App User Experience: An Analysis of Education App User Reviews. In *2024 4th International Conference on Advanced Research in Computing (ICARC)*. 223–228. https://doi.org/10.1109/ICARC61713.2024.10499727

[6] Maram Assi, Safwat Hassan, Stefanos Georgiou, and Ying Zou. 2023. Predicting the Change Impact of Resolving Defects by Leveraging the Topics of Issue Reports in Open Source Software Systems. *ACM Trans. Softw. Eng. Methodol.* 32, 6, Article 141 (sep 2023), 34 pages. https://doi.org/10.1145/3593802

[7] Maram Assi, Safwat Hassan, Yuan Tian, and Ying Zou. 2021. FeatCompare: Feature comparison for competing mobile apps leveraging user reviews. *Empirical Software Engineering* 26, 5 (2021).

[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *J. Mach. Learn. Res.* 3 (2003), 1137–1155. https://api.semanticscholar.org/CorpusID:221275765

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. Curran Associates Inc., Red Hook, NY, USA.

[10] Published by Laura Ceci and Mar 4. 2024. Messaging apps: Most popular by global downloads 2024. https://www.statista.com/statistics/1263360/most-popular-messenger-apps-worldwide-by-monthly-downloads/

[11] Laura Ceci. 2024. Number of mobile app downloads worldwide from 2016 to 2023. https://www.https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/. (Last accessed May 2024).

[12] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*. 767–778.

[13] Jacob Cohen. 1960. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement* 20, 1 (1960), 37–46.

[14] Fabiano Dalpiaz and Micaela Parente. 2019. RE-SWOT: From User Feedback to Requirements via Competitor Analysis. In *Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ '19, Vol. 11412)*. 55–70.

[15] Evan DeFilippis, Stephen Michael Impink, Madison Singell, Jeffrey T Polzer, and Raffaella Sadun. 2022. The impact of COVID-19 on digital communication patterns. *Humanities and Social Sciences Communications* 9, 1 (2022).

[16] Peter Devine, James Tizard, Hechen Wang, Yun Sing Koh, and Kelly Blincoe. 2022. What's Inside a Cluster of Software User Feedback: A Study of Characterisation Methods. In *2022 IEEE 30th International Requirements Engineering Conference (RE)*. 189–200. https://doi.org/10.1109/RE54965.2022.00023

[17] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A Survey on In-context Learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1107–1128. https://doi.org/10.18653/v1/2024.emnlp-main.64

[18] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration Code Generation via ChatGPT. arXiv:2304.07590 [cs.SE]

[19] Paulo Sérgio Henrique Dos Santos, Alberto Dumont Alves Oliveira, Thais Bonjorni Nobre De Jesus, Wajdi Aljedaani, and Marcelo Medeiros Eler. 2024. Evolution may come with a price: analyzing user reviews to understand the impact of updates on mobile apps accessibility. In *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems* (<conf-loc>, <city>Maceió</city>, <country>Brazil</country>, </conf-loc>) *(IHC '23)*. Association for Computing Machinery, New York, NY, USA, Article 52, 11 pages. https://doi.org/10.1145/3638067.3638081

[20] Hugging Face. 2024. Mixtral-8x7B-Instruct-v0.1. https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1. Accessed: 2024-03-28.

[21] Zhiyu Fan, Xiang Gao, Abhik Roychoudhury, and Shin Hwei Tan. 2022. Improving automatically generated code from Codex via Automated Program Repair. *ArXiv* abs/2205.10583 (2022). https://api.semanticscholar.org/CorpusID:248986410

[22] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 2013. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. 1276–1284.

[23] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 2013. Why people hate your app: making sense of user feedback in a mobile app store. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 1276–1284. https://doi.org/10.1145/2487575.2488202

[24] Cuiyun Gao, Yaoxian Li, Shuhan Qi, Yang Liu, Xuan Wang, Zibin Zheng, and Qing Liao. 2023. Listening to Users' Voice: Automatic Summarization of Helpful App Reviews. *IEEE Transactions on Reliability* 72, 4 (2023), 1619–1631. https://doi.org/10.1109/TR.2022.3217566

[25] Cuiyun Gao, Wujie Zheng, Yuetang Deng, David Lo, Jichuan Zeng, Michael R. Lyu, and Irwin King. 2019. Emerging App Issue Identification from User Feedback: Experience on WeChat. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 279–288. https://doi.org/10.1109/ICSE-SEIP.2019.00040

[26] Shanquan Gao, Lei Liu, Yuzhou Liu, Huaxiao Liu, and Yihui Wang. 2020. Updating the goal model with user reviews for the evolution of an app. *J. Softw. Evol. Process* 32, 8 (Aug. 2020), 24 pages. https://doi.org/10.1002/smr.2257

[27] Safwat Hassan, Heng Li, and Ahmed E. Hassan. 2022. On the Importance of Performing App Analysis Within Peer Groups. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 890–901. https://doi.org/10.1109/SANER53432.2022.00107

[28] Ruidan He, Wee Sun Lee, Hwee Tou Ng, and Daniel Dahlmeier. 2017. An Unsupervised Neural Attention Model for Aspect Extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL '17)*. 388–397.

[29] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE]

[30] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685 [cs.CL]

[31] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. arXiv:2311.05232 [cs.CL]

[32] Claudia Iacob and Rachel Harrison. 2013. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. 41–44.

[33] T. Johann, C. Stanik, A. M. A. B., and W. Maalej. 2017. SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews. In *Proceedings of the 25th International Requirements Engineering Conference (RE '17)*. 21–30.

[34] Yuanchun Li, Baoxiong Jia, Yao Guo, and Xiangqun Chen. 2017. Mining User Reviews for Mobile App Comparisons. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (Sept. 2017), 75:1–75:15.

[35] Vitor Lima, Jacson Barbosa, and Ricardo Marcacini. 2023. Learning Risk Factors from App Reviews: A Large Language Model Approach for Risk Matrix Construction. https://doi.org/10.21203/rs.3.rs-3182322/v1

[36] Huaxiao Liu, Yihui Wang, Yuzhou Liu, and Shanquan Gao. 2021. Supporting features updating of apps by analyzing similar products in App stores. *Information Sciences* 580 (2021), 129–151. https://doi.org/10.1016/j.ins.2021.08.050

[37] Z. Liu, C. Chen, J. Wang, M. Chen, B. Wu, Z. Tian, Y. Huang, J. Hu, and Q. Wang. 2024. Testing the Limits: Unusual Text Inputs Generation for Mobile App Crash Detection with Large Language Model. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1685–1696. https://doi.ieeecomputersociety.org/

[38] Walid Maalej and Hadeer Nabil. 2015. Bug report, feature request, or simply praise? On automatically classifying app reviews. In *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. 116–125. https://doi.org/10.1109/RE.2015.7320414

[39] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented Language Models: a Survey. arXiv:2302.07842 [cs.CL]

[40] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).

[41] P. Mishra, U. Singh, C. M. Pandey, P. Mishra, and G. Pandey. 2019. Application of student's t-test, analysis of variance, and covariance. *Annals of Cardiac Anaesthesia* 22, 4 (October–December 2019), 407–411. https://doi.org/10.4103/aca.ACA_94_19

[42] Quim Motger, Xavier Franch, Vincenzo Gervasi, and Jordi Marco. 2024. Unveiling Competition Dynamics in Mobile App Markets Through User Reviews. In *Requirements Engineering: Foundation for Software Quality*, Daniel Mendez and Ana Moreira (Eds.). Springer Nature Switzerland, Cham, 251–266.

[43] OpenAI. 2024. GPT-4o mini. https://platform.openai.com/docs/

[44] Kate O'Flaherty. 2020. Zoom beats Microsoft Teams, google meet with game-changing new features. https://www.forbes.com/sites/kateoflahertyuk/2020/10/14/zoom-beats-microsoft-teams-google-meet-with-game-changing-new-features/

[45] D. Pagano and W. Maalej. 2013. User feedback in the appstore: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*. 125–134. https://doi.org/10.1109/RE.2013.6636712

[46] Juan Ramos. 2003. Using TF-IDF to determine word relevance in document queries. In *Proceedings of the 1st instructional Conference on Machine Learning (iCML '03)*. 1–4.

[47] Christian Rigg and Nikshep Myle. 2021. Zoom Video Conferencing Service Review. https://www.techradar.com/reviews/zoom#section-zoom-features

[48] Konstantinos I. Roumeliotis, Nikolaos D. Tselikas, and Dimitrios K. Nasiopoulos. 2024. LLMs in e-commerce: A comparative analysis of GPT and LLaMA models in product review evaluation. *Natural Language Processing Journal* 6 (2024), 100056. https://doi.org/10.1016/j.nlp.2024.100056

[49] Mario Sänger, Ulf Leser, Steffen Kemmerer, Peter Adolphs, and Roman Klinger. 2016. SCARE — The Sentiment Corpus of App Reviews with Fine-grained Annotations in German. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA), Portorož, Slovenia, 1114–1121. https://aclanthology.org/L16-1178

[50] Simone Scalabrino, Gabriele Bavota, Barbara Russo, Massimiliano Di Penta, and Rocco Oliveto. 2019. Listening to the Crowd for the Release Planning of Mobile Apps. *IEEE Transactions on Software Engineering* 45, 1 (2019), 68–86. https://doi.org/10.1109/TSE.2017.2759112

[51] Faiz Ali Shah, Yevhenii Sabanin, and Dietmar Pfahl. 2016. Feature-based evaluation of competing apps. In *Proceedings of the ACM International Workshop on App Market Analytics (WAMA '16)*. 15–21.

[52] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. 2018. The Impact of Annotation Guidelines and Annotated Data on Extracting App Features from App Reviews. *CoRR* abs/1810.05187 (2018).

[53] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. 2019. Is the SAFE Approach Too Simple for App Feature Extraction? A Replication Study. In *Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 19)*. 21–36.

[54] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. 2019. Using app reviews for competitive analysis: tool support. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics (WAMA '19)*. 40–46.

[55] Chris Stokel-Walker. 2020. How Skype lost its crown to Zoom. https://www.wired.com/story/skype-coronavirus-pandemic/

[56] H. Strobelt, A. Webson, V. Sanh, B. Hoover, J. Beyer, H. Pfister, and A. M. Rush. 2023. Interactive and Visual Prompt Engineering for Ad-hoc Task Adaptation with Large Language Models. *IEEE Transactions on Visualization and Computer Graphics* 29, 01 (jan 2023), 1146–1156. https://doi.org/10.1109/TVCG.2022.3209479

[57] Yanqi Su, Yongchao Wang, and Wenhua Yang. 2019. Mining and Comparing User Reviews across Similar Mobile Apps. In *2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*. 338–342. https://doi.org/10.1109/MSN48538.2019.00070

[58] M. Tushev, F. Ebrahimi, and A. Mahmoud. 2022. Domain-Specific Analysis of Mobile App Reviews Using Keyword-Assisted Topic Models. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 762–773. https://doi.org/10.1145/3510003.3510201

[59] Phong Minh Vu, Tam The Nguyen, Hung Viet Pham, and Tung Thanh Nguyen. 2015. Mining User Opinions in Mobile App Reviews: A Keyword-Based Approach (T). In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE '15)*. 749–759.

[60] Yihui Wang, Shanquan Gao, Xingtong Li, Lei Liu, and Huaxiao Liu. 2022. Missing standard features compared with similar apps? A feature recommendation method based on the knowledge from user interface. *Journal of Systems and*

Software 193 (2022), 111435. https://doi.org/10.1016/j.jss.2022.111435

[61] Yihui Wang, Shanquan Gao, Yan Zhang, Huaxiao Liu, and Yiran Cao. 2022. UISMiner: Mining UI suggestions from user reviews. *Expert Systems with Applications* 208 (2022), 118095. https://doi.org/10.1016/j.eswa.2022.118095

[62] Jialiang Wei. 2023. Enhancing Requirements Elicitation through App Stores Mining: Health Monitoring App Case Study. In *2023 IEEE 31st International Requirements Engineering Conference (RE)*. IEEE, 396–400.

[63] Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Binbin Xu, Pierre Louis Bernard, and Gérard Dray. 2023. Zero-shot Bilingual App Reviews Mining with Large Language Models. In *2023 IEEE 35th International Conference on Tools with Artificial Intelligence (ICTAI)*. 898–904. https://doi.org/10.1109/ICTAI59109.2023.00135

[64] Xiancai Xu, Jia-Dong Zhang, Rongchang Xiao, and Lei Xiong. 2023. The Limits of ChatGPT in Extracting Aspect-Category-Opinion-Sentiment Quadruples: A Comparative Analysis. arXiv:2310.06502 [cs.CL]

[65] Pengyu Xue, Linhao Wu, Zhongxing Yu, Zhi Jin, Zhen Yang, Xinyi Li, Zhenyu Yang, and Yue Tan. 2024. Automated Commit Message Generation with Large Language Models: An Empirical Study and Beyond. *arXiv preprint arXiv:2404.14824* (2024).

[66] Wenxuan Zhang, Yue Deng, Bing Liu, Sinno Jialin Pan, and Lidong Bing. 2023. Sentiment Analysis in the Era of Large Language Models: A Reality Check. arXiv:2305.15005 [cs.CL]

[67] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

[68] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large Language Models are Human-Level Prompt Engineers. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=92gvk82DE-